Tasohyppelypeli Godot-pelimoottorilla

Vertailu Godot- ja Unity-pelimoottorin välillä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeenlinnan korkeakoulukeskus, Tietojenkäsittelyn koulutus

2019

Jarkko Eloranta

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus
Hämeenlinnan korkeakoulukeskus

Tekijä	Jarkko Eloranta	**Vuosi** 2019
Työn nimi	Tasohyppelipeli Godot-pelimoottorilla	
Työn ohjaaja/t	Tommi Lahti	

TIIVISTELMÄ

Työn tavoitteena oli tutkia Godot-pelimoottorin pääpiirteitä ja toimintoja ja vertailla niitä Unity-pelimoottorin kanssa sekä luoda 2D-tasohyppelypeli Godotilla. Työn aihe syntyi kiinnostuksesta etsiä vaihtoehto Unity-pelimoottorille, joka on suosittu ja tunnettu pelinkehityksessä. Godot valittiin vaihtoehdoksi sen lupaamien erityisominaisuuksien kannalta.

Teoriaosuudessa tutkittiin Godotin ominaisuuksia, käyttötapauksia ja sen käyttämistä. Lisäksi vertailtiin sen lisenssiehtoja, toimintoja ja ominaisuuksia Unity-pelimoottorin lisenssien ja toimintojen kanssa.

Käytännön osana luotiin 2D-tasohyppelypeli, johon rakennettiin 2 pelitasoa sekä liikkumis-, hyppäämis- ja hyökkäyskontrollit ja animaation kytkeminen näihin. Tämän lisäksi kuvattiin kahden eri vihollishahmon luonti ja tekoälyn hyödyntäminen niissä. Menetelmänä käytettiin toiminnallista opinnäytetyötä.

Työssä tuli ilmi, että Godot ja Unity soveltuvat eri tarkoituksiin pelinkehityksessä. Godot esimerkiksi tukee suurempaa määrää ohjelmointikieliä kuin Unity, mutta Unitylle on saatavilla enemmän materiaaleja sen oppimiseen. Lisäksi augmented reality- eli lisätyn todellisuuden tuki on Godotissa erittäin rajattu verrattuna Unityyn. Käytännön osuudessa toisaalta esiteltiin Godotin toimintoja peliä luodessa, joista tuli tietoon, miten toteutetaan tai muodostetaan haluttu toiminto Godotin muokkausohjelmassa, sekä kuinka AStar- ja Navigation2D-reitinhakualgoritmeja käytetään.

Avainsanat Godot, tasohyppely, Unity, tekoäly

Sivut 36 sivua, joista liitteitä 1 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Hämeenlinna University Centre

Author Jarkko Eloranta **Year** 2019

Subject Platform game on Godot-game engine

Supervisors Tommi Lahti

ABSTRACT

The goal of this thesis was to examine Godot game engine's main features and functions and compare them with those of the Unity engine. The goal also includes creating a 2D platformer game with Godot. The subject of this work originated from an interest in finding an alternative to Unity game engine, which is a widely known and used game engine. Godot was chosen because of the special features it promises.

The theory part researches and explores Godot's features, use cases and its usage overall. Additionally, Godot's license terms, functions and features are compared with those of the Unity engine.

The practical part of this thesis describes the creation of move, jump and attack controls for a 2D platform game and how to apply animations to those controls. In addition, this work documents the creation of two different enemy characters and the use of artificial intelligence in them. This is a functional thesis.

The work revealed that both Godot and Unity engines suit different purposes in game development. While Godot supports a higher number of programming languages, Unity has more material and tutorials to teach its usage. The support of augmented reality is also extremely limited in Godot compared with Unity's support. On the other hand, the functional part of this thesis presents Godot's functions while creating a game that also helps to teach how a certain functionality is made in Godot's editor, and how AStar and Navigation2D pathfinding algorithms are used.

Keywords Godot, platform game, Unity, AI

Pages 36 pages including appendices 1 pages

SISÄLLYS

Liitteet

Liite 1 Godotin ja Unityn erot

1 JOHDANTO

Tämän työn aihe valittiin halusta löytää vaihtoehto Unity-pelimoottorille. Unreal Engine-pelimoottorin ja Godotin väliltä valittiin Godot, sillä se on uudempi ja tuntemattomampi pelimoottori verrattuna Unreal Engine-pelimoottoriin. Godot valittiin aiheeksi täten, sillä sen listatut erityisominaisuudet vaikuttivat kiinnostavilta ja tutkinnan arvoisilta.

Tämän opinnäytetyön tavoitteena on tutkia Godot-pelimoottoria ja vertailla sen pääpiirteitä, erityisominaisuuksia ja muokkausohjelmaa Unity-pelimoottorin kanssa. Lisäksi kuvataan, kuinka Godotia ja sen päätoimintoja käytetään. Tämä työ on suunnattu Godot-pelimoottorista ja pelinkehityksestä kiinnostuneille, jotka haluavat etsiä erilaisia vaihtoehtoja Unity-pelimoottorille.

Tämän lisäksi toiminnallisena osana luodaan tasohyppelypeli Godot-pelimoottorilla, jonka avulla kuvataan mahdollisia eroja Unity-pelimoottorin kanssa. Rajoituksena pelissä luodaan liikkumis- ja hyökkäyskontrollit, pelin aloitus- ja lopputasot sekä alkukantainen tekoäly kahdelle viholliselle.

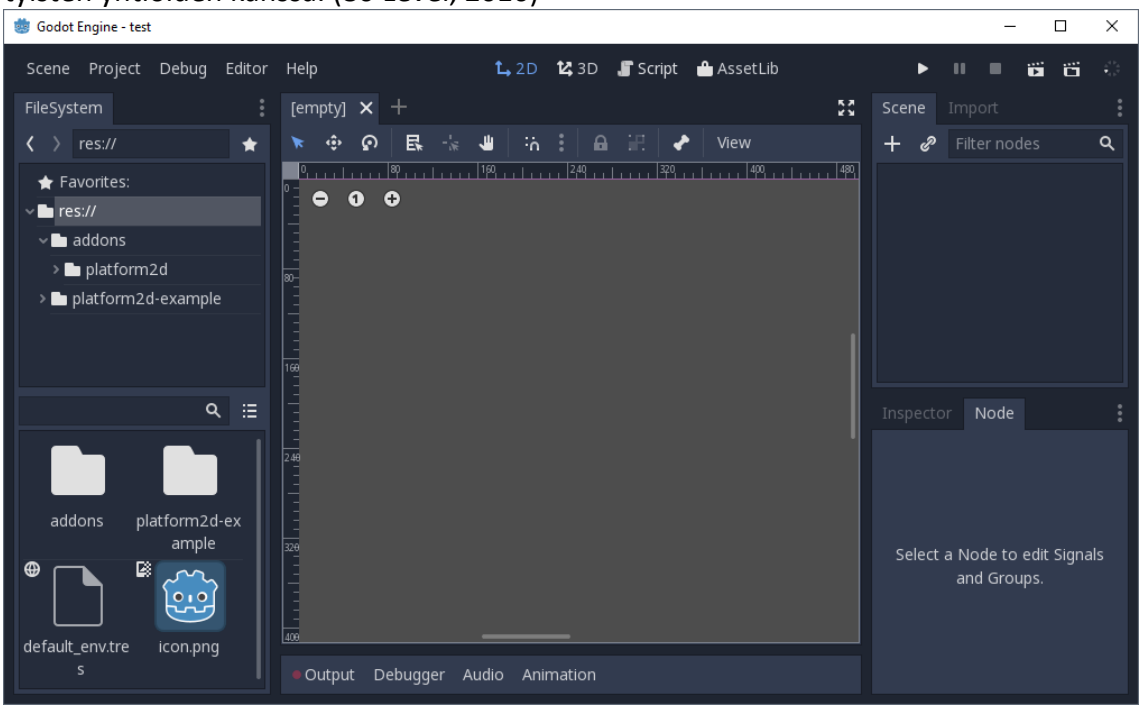
Tämä opinnäytetyö vastaa seuraaviin kysymyksiin:

- Mikä on Godot-pelimoottori?
- Miten Godot- ja Unity-pelimoottorit eroavat toisistaan?
- Miten liikkumis- ja hyökkäyskontrollit luodaan?
- Miten pelitasot luodaan?
- Miten tekoäly luodaan ja miten sitä hyödynnetään?

2 MIKÄ ON GODOT-PELIMOOTTORI?

Godot on alustariippumaton ilmainen pelimoottori, joka tukee molempia 2D- ja 3D-ympäristöjä. Godotin ohjelmointirajapinta on olio-painotteinen (Godot, n.d. c). Godot käyttää Bullet-fysiikkamoottoria, jota käyttää myös 3D-grafiikan mallinnusohjelma Blender (Blender Foundation, n.d.).

Godot on yhteisön kehittämä pelimoottori, jonka takana ei ole yhtiötä, yritystä tai liikettä. Ariel Manzur ja Juan Linietsky kuitenkin omistavat Godotin. Square Enix, Daedalic ja jopa Sony ovat käyttäneet Godotia sen yksityisten yhtiöiden kanssa. (80 Level, 2016)



Kuva 1. Godotin editori 2D-näkymässä.

Godot-yhteisö ei ole ainoa, joka tukee Godotia. Godotia edesauttaa myös Software Freedom Conservancy-hyväntekeväisyys, joka auttaa avoimen lähdekoodin projekteja, kuten Godotia ja Git-versionhallintajärjestelmää voittoa tavoittelematta (Software Freedom Conservancy, n.d.).

Godot tarjoaa avoimen lähdekoodin MIT-lisenssin alla. Tämä antaa vapauden käyttää Godottia mihin tarkoitukseen tahansa, sekä muokata ja levittää sitä kaupallisesti toisen lisenssin alla. Ainoa rajoite kaupalliseen levitykseen toisen lisenssin alla on se, että alkuperäinen tekijänoikeushuomautus ja lisenssilausunto on annettu eteenpäin levityksen yhteydessä. Tämä tarkoittaa sitä, että vaikka johdannaisessa tuotteessa saattaa olla erilainen lisenssi, tulisi sen silti ilmoittaa sen dokumentaatiossa, että se juontaa MIT-lisenssöidystä Godot-pelimoottorista. Lisenssiehdot ja tekijänoikeudet eivät kuitenkaan päde sisältöön, mitä sillä luodaan. Godotilla luodut pelit voidaan lisenssoida tekijän parhaaksi näkemällä tavalla. On kuitenkin hyvä

huomata, että Godotilla luotujen pelien dokumentaatiossa tai lopputeksteissä pitää sisältyä linkki Godotin lisenssiin (godotengine.org/license), kuten Godotin lisenssissä on määritelty. Godot myös käyttää useita kolmannen osapuolen ohjelmakirjastoja ja koodin pätkiä, jotka ovat luokiteltu niiden omilla lisensseillä ja tekijänoikeushuomautuksilla. Kaikki nämä komponentit ovat yhteensopivia Godotin MIT-lisenssin kanssa. (Godot, n.d. e).

Godotilla on tehty lukuisia pelejä, joista ehkä yksi tunnetuimmista on OKAM Studion kehittämä The Interactive Adventures of Dog Mendonça & Pizzaboy, joka perustuu Filipe Melo:n sarjakuva trilogiaan The Adventures of Dog Mendonça & Pizzaboy. The Interactive Adventures of Dog Mendonça & Pizzaboy-pelin kehitys alkoi kuuden henkilön tiimillä, joka oli taitava taiteen ja animoinnin suhteen, mutta aloittelija pelinkehityksessä. (Suckley, 2016)

Kuva 2. Kuva seikkailupelistä The Interactive Adventures of Dog Mendonça & Pizzaboy, joka on toteutettu Escoria-seikkailupeli viitekehyksellä Godotin editoria mukauttaen.

Godotin lisäksi tiimi käytti erityisesti Escoria-seikkailupeli viitekehystä. Keskeisenä konseptina tämän viitekehyksen käytössä on se, että jokainen tiimin jäsen, erityisesti pelin suunnittelijat ja artistit, pystyvät käyttämään omia taitojaan mahdollisimman hyvin ilman jatkuvaa avustusta ohjelmoijan kanssa toteutuksessa. (Suckley, 2016) Tämä ei kuitenkaan tarkoita sitä, ettei pelinteko Escorialla vaatisi ollenkaan ohjelmointia, tai että se olisi suljettu tuote (Manzur, 2016).

Godot voi toimia myös opetusvälineenä. Yhdysvaltojen osavaltio Länsi-Virginia otti käyttöönsä lukuvuonna 2016 ensimmäisen kerran ohjelmointi, sovellus- ja pelinsuunnittelu opetussuunnitelman, jossa tarkoituksena on käyttää Godotia. Trixie Devine, opettaja Graftonin lukiosta ja yksi tämän opetussuunnitelman edistäjistä kertoo, että alun perin suunnitelmana oli käyttää Microsoftin Project Spark-pelinluonti videopeliä, joka lakkautettiin. Tämän takia osavaltio joutui etsimään vaihtoehtoja, joista Godot oli seuraava. Devine kertoo, että löytäessään Godotin hän rakastui siihen, ja

jos hänen olisi pitänyt valita joko Project Sparkin ja Godotin välillä, hän olisi valinnut Godotin. (Brasseur, 2016)

Godotin valinta opetussuunnitelmaan oli myös Godotin runsaan oppimateriaalin ansiota. Rajapintaviitteet, kurssit ja lukuisat muut materiaalit ovat kaikki saatavilla Godotin virallisissa dokumentaatioissa. Godot tarjoaa tämän lisäksi myös virallisia oppivideoita YouTuben kautta, ja StackOverflow'n kaltaisella kysymys-ja-vastaus-sivulla ja Freenode IRC-kanavan kautta. Devinen mukaan Godotin foorumit ovat myös erittäin tärkeät. (Brasseur, 2016)

Länsi-Virginian osavaltio ei ole kuitenkaan ensimmäinen, joka on käyttänyt Godotia opetusympäristössä. Team Krishna, ensimmäinen tiimi, joka osallistui Global Learning XPRIZE-kilpailuun, jonka tavoitteena on hävittää lukutaidottomuus, käytti ja edisti avoimen lähdekoodin työkaluja kuten Godotia, sillä se tekee ohjelmistoja, joilla opetetaan lapsia lukemaan käyttämällä koneoppimista ja pelillistämistä. (Brasseur, 2016)

3 VERTAILU UNITYN JA GODOTIN VÄLILLÄ

Unity ja Godot ovat molemmat ilmaisia pelimoottoreita. Niillä on kuitenkin erilaisia pääpiirteitä, erityisominaisuuksia ja lisenssejä, joita kuvaillaan ja vertaillaan tulevissa kappaleissa.

3.1 Vertailu Unityn ja Godotin pääpiirteiden välillä

Unity ja Godot tukevat molempia 2D- ja 3D-ympäristöjä, sekä molempien muokkausohjelmat tukevat Windows-, Mac-, ja Linux-sovellusalustoja. Godot toimii myös FreeBSD-, OpenBSD- ja Haiku-sovellusalustoilla. Unity kuitenkin on riippuvainen erilaisista paketeista ja ohjelmistokirjastoista toimiakseen Linux-sovellusalustalla (natosha-bard, 2015).

Godot siis toimii suuremmissa määrissä sovellusalustoja kuin Unity. Godot ei myöskään ole riippuvainen erillisistä paketeista kääntämisen yhteydessä. (Godot, n.d. b)

Unity tarjoaa suurimman valikoiman alustoja, joita se tukee; yli 25 alustaa mukaan lukien puhelimet, tietokoneet, pelikonsolit, VR, AR ja verkko (Unity Technologies, n.d. b). Godot toisaalta tukee puhelinalustoista iOS- ja Android-alustoja, tietokone alustoista Windows, macOS, Linux, UWP, *BSD ja Haiku ja verkkoalustoista HTML ja Web Assembly (Godot, n.d. b).

Godot on suunniteltu alusta lähtien kitkattomaan tiimityöskentelyyn. Sen tiedostojärjestelmän käyttö toimii hyvin versionhallintajärjestelmien kuten Git ja Mercurial kanssa. Tämän lisäksi Godotilla on mahdollisuus hyödyntää kohtausten erillistämistä, jolla jokainen tiimijäsen voi työskennellä oman kohtauksen tai ”scene” parissa muokkaamatta muiden tiimijäsenten scenejä. (Godot, n.d. b) Scene on tietty osa pelistä, joka näkyy halutulla hetkellä. Esimerkiksi pelin aloitusruutu on yksi scene, aloitustaso toinen ja niin edelleen.

Unity toisaalta käyttää tiimityöskentelyyn Unity Collaborate-toimivuutta, jossa on mahdollista jakaa ja synkronoida projektit pilvivaraston avulla. Tämä toimivuus tekee projektin jakamisen helpommaksi, sillä se ei vaadi ulkoisia versionhallintajärjestelmiä, kuten Godot. (Unity Technologies, n.d. b)

Tiimityöskentelyyn Unity vaikuttaa toimivammalta ratkaisulta kuin Godot ainakin pieniin projekteihin, sillä se ei vaadi versionhallintajärjestelmien käyttämistä tai osaamista, jos versionhallintajärjestelmät eivät ole tuttuja tiimille. Versionhallintajärjestelmät kuitenkin tekevät suuremman projektin tekemisestä helpompaa, sillä muutokset ja lisäykset tulevat helpommin esille niissä kuin Unity Collaborate-ilmaisversiossa. Tämä ilmaisversio lisäksi antaa mahdollisuuden vain enintään kolmelle tiimijäsenelle ja 1GB pilvivarastointia (Unity Technologies, n.d. d).

3.2 **Vertailu Unityn ja Godotin erityisominaisuuksien välillä**

Erityisominaisuuksien välillä Unitylla ja Godotilla on erittäin paljon eroavaisuuksia. Muun muassa Unitylla on ulkoinen koodinmuokkausohjelma, mutta Godotin koodinmuokkausohjelma on integroitu itse editoriin. Godotin dokumentointi on myös integroitu editoriin, toisin kuin Unityn dokumentointi löytyy ainoastaan Unity Technologies-dokumentaatio verkkosivuilta. Tämä tarkoittaa sitä, että Godotin dokumentaatio on aina saatavilla Internet-yhteydestä tai verkkosivupalvelimien tilasta riippumatta.

AR tai Augmented Reality, suomeksi lisätty todellisuus, on täysin tuettu Unityn editorissa. Unity tukee muun muassa Vuforia-, Google ARCore- ja Apple ARKit-alustoja. (Unity Technologies, n.d. b). Godot toisaalta tukee ainoastaan Apple ARKit-alustaa. Apple ARKit ei kuitenkaan ole osa Godotin ydintä (Bastiaan, 2017).

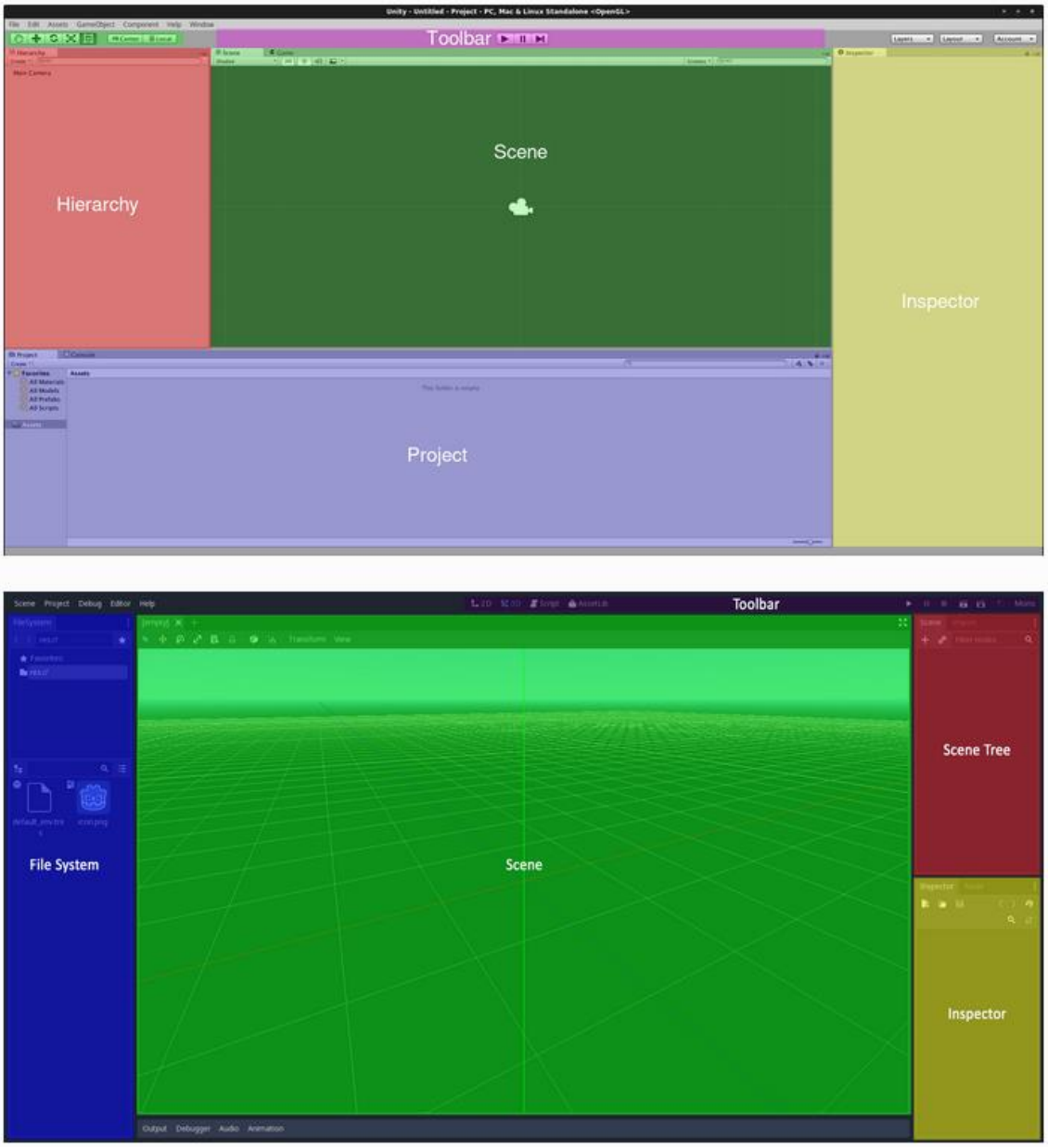
Asennuskoko on Godotilla on erittäin pieni: vain noin 20 megatavua (Godot, n.d.a). Unityn koko toisaalta on version mukaan noin 2-3 gigatavua. Asennuskokojen ero on erittäin suuri; Godot on erittäin helppo ottaa käyttöön nopeasti pienen asennuskoon perusteella.

Godot on erittäin suopea MIT-lisenssinsä suhteen, kuten aikaisemmin kuvattiin. Unitylla on toisaalta lisenssejä erilaisiin käyttötarkoituksiin. Esimerkiksi Personal-lisenssi Unitylla on ilmainen, mutta vaatii ettei projektin rahoitus tai liikevaihto ylitä yli 100 000 dollaria vuodessa Personal-lisenssillä ei voi poistaa Unityn aloitusruutua. Unity Plus-lisenssi toisaalta maksaa 25 dollaria kuukaudessa ja sallii muun muassa aloitusruudun muokkaamisen tai poistamisen ja mainosten ja sovelluksen sisäisten ostosten tekemisen. Unitylla on myös mahdollisuus Unity Pro-lisenssiin, joka maksaa 125 dollaria kuukaudessa ja joka tarjoaa esimerkiksi suoran opastuksen Unityn työntekijöiltä, lisää Unity Collaborate-toimivuuksia ja tarjoaa ilmaisia taidepaketteja ja niin edelleen. (Unity Technologies, n.d. c).

Godotilla on erityisominaisuus, jota Unitylla ei ole: Pysyvä suora editointi. Tämä tarkoittaa sitä, että editorin pelitilassa tehdyt muutokset ovat pysyviä. Tämä helpottaa testaamista, sillä pelitilassa scene saattaa toimia eri tavoin kuin sen ulkopuolella. (Godot, n.d. b)

3.3 **Vertailu Unityn ja Godotin editorien välillä**

Godot on luonut oppaan, joka antaa yleiskatsauksen Godot-pelimoottoriin Unity-pelimoottorin käyttäjän näkökulmasta. Liittestä 1 nähdään kuva taulukosta, jossa Godot tuo Unityn ja Godotin lisenssien, sovellusympäristöjen, Scene-järjestelmien, erityispiirteiden ja kolmansien osapuolten työkalujen erot.



Kuva 3. Godotin ja Unityn editorien erot (Godot, n.d. c).

Kuvassa 3 kuvataan Godotin ja Unityn editorien eroja. Ylemmässä näkymässä on Unityn-editori ja alemmassa Godotin-editori. Molemmat ovat perusnäkymässä ilman muokkauksia. Nopeasti katsottuna näiden kahden pelimoottorien näkymät näyttävät samankaltaisilta. Ne kuitenkin eroavat rakenteeltaan toisistaan melko paljon, vaikka tarjoavatkin samankaltaisia toimintoja. Godotilla on kuitenkin vain yksi asetustiedosto, eikä ollenkaan metadataa. Tämä tarkoittaa sitä, että Godot on paljon helpompi käyttää versionhallintajärjestelmien kuten Git kanssa. (Godot, n.d. b)

Yhden asetustiedoston sekä metadatan olemattomuuden lisäksi Godotin Scene-paneeli on samankaltainen kuin Unityn Hierarchy-paneeli, mutta pienillä eroavaisuuksilla. Godotin jokaisella nodella tai solmulla on tietty toiminto, joka tekee Godotin lähestymistavasta visuaalisesti

kuvaavamman, eli silmäyksellä voi nähdä helpommin, mitä jokainen scene tekee (Godot, n.d. b).

Scene-paneelin lisäksi Godotin Inspector-paneeli on suunniteltu näyttävän ainoastaan ominaisuuksia. Tämän ansiosta oliot voivat tuoda suuren määrän parametreja käyttäjälle kätkemättä toimintoja ohjelmointikielen sovellusrajapinnoille. Lisäksi, Godot sallii näiden ominaisuuksien muokkaamisen visuaalisesti reaaliajassa ilman ohjelmointia (Godot, n.d. b).

Godotin editori on vakaampi kuin Unityn, sillä Godotin editori ja peli pyörivät erillisissä ikkunoissa. Tämä myös nopeuttaa projektien avaamista, sulkemista ja ajamista Unityn editoriin verrattuna. Erilliset ikkunat editorille ja pelille Godotissa sallii myös reaaliaikaisen muokkaamisen, joka synkronoi editorissa tehdyt muutokset peli-ikkunan kanssa. Tämä mahdollistaa pelitasojen luonnin pelaamisen yhteydessä. Kaiken tämän lisäksi Godotilla on mahdollisuus etätestaukseen puhelimella, tabletilla tai jopa selaimella, jossa reaaliaikainen muokkaus ja testaus on myös mahdollista. (Godot, n.d. b)

3.4 **Vertailu Unityn ja Godotin tuettujen ohjelmointikielien välillä**

Unity tukee C#-ohjelmointikieltä natiivisti. Se tukee myös muita .NET-ohjelmointikieliä, jos ne voivat kääntää kelvollisen DLL:n (Dynamic-Link Library englanniksi). (Unity Technologies, n.d. a).

JavaScript-kielen kaltainen Unityn oma UnityScript oli aikaisemmin tuettu, mutta tuki lakkautettiin vuonna 2017, sillä vain pieni murto-osa projekteista käytti UnityScriptiä C#:n sijasta. Yksi monista syistä, miksi UnityScript lakkautettiin, oli myös C#-kurssien ja työkalujen määrä, mitä on olemassa. (Fine, 2017)

Godot toisaalta tukee Unityn C#-ohjelmointikielen lisäksi GDScriptiä, joka on Godotin oma Pythonin kaltainen kieli, C++-ohjelmointikieltä sekä Godot-yhteisön tekemät tuet Pythonille, Nim:ille, D:lle ja muille. Godotin Mono-versio, joka tukee C#-ohjelmointikieltä, vaatii Mono ohjelmistokehityspaketti version 5.12.0 toimiakseen (Godot, n.d. a). Godot ei kuitenkaan suosittele Mono versiota tuotantoon, sillä C#-tuki on vasta alfa vaiheessa (Etcheverry 2017).

Tiivistettynä, Godot tarjoaa enemmän ohjelmointikieli mahdollisuuksia, kuin Unity. Godotin dokumentaatio ei kuitenkaan ole täyteläinen kaikille ohjelmointikielille, mitä se tukee. Unityn C#-tuki ja dokumentointi on erittäin kattava.

3.5 Vertailu Unityn suljetun- ja Godotin avoimen-lähdekoodin välillä

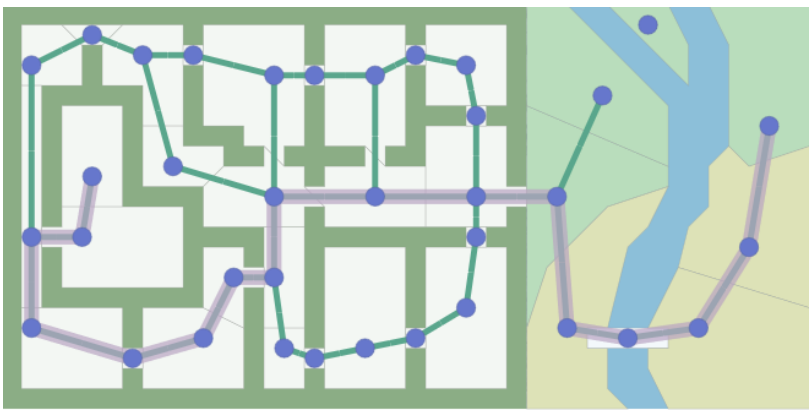
Godotilla on helmikuusta 2014 ollut täysin avoin lähdekoodi, kun taas Unityn C#-lähdekoodi on saatavilla vain viittauksiin. Tämä tarkoittaa sitä, että Unityn lähdekoodin lisenssi tarjoaa oikeudet vain lähdekoodin lukemiseen, mutta ei muokkaamiseen. Syy, miksi Unity Technologies tarjoaa lähdekoodin lukemisen, on muutosten tarkastelu eri versioiden välillä. (Pranckevičius, 2018) Godotin kehitys on yhteisöpainotteinen; se antaa valtuudet käyttäjillensä muokata Godotia tarpeidensa mukaan. (Godot, n.d. d)

Kaiken kaikkiaan lähdekoodi saatavuuksien eroavaisuudet Unitylla ja Godotilla ja syyt niiden saatavuuteen ovat täysin erilaiset. Täysin aloittelijalle pelin tekemisessä ei lähdekoodin saatavuudelle ole juurikaan käyttöä kummassakaan tapauksessa. Kokeneemmalle käyttäjälle voi lähdekoodin tarkastelusta olla kuitenkin hyötyä. Esimerkiksi vanhempaan Unity-versioon tottunut pelinkehittäjä päättää päivittää uudempaan versioon lukiessaan lähdekoodista uusien toimintojen lisäyksen. Samankaltainen ajatus pätee myös Godotiin; kokeneempi käyttäjä voi muokata Godotia lisäämällä haluamansa toiminnon.

3.6 Vertailu Unityn ja Godotin sisäänrakennetun tekoälyn välillä

Godotilla ja Unityllä on pari erilaista mahdollisuutta polunetsintään. Godot tukee muun muassa A*- tai AStar-algoritmia ja sen sisäänrakennettua noodia. Tämän lisäksi se tukee myös Navigation2D-noodia, joka on myös sisäänrakennettu Godotiin. Navigation2D-reitinetsintää on myös mahdollista hyödyntää Unity projekteissa, mutta tällä hetkellä se on maksullinen Unityssä (Unity Technologies, n.d. e).

AStar-algoritmi tarvitsee syöttötiedoksi diagrammin, joka koostuu sijainneista sekä niiden välisistä yhteyksistä. AStar tunnistaa ainoastaan syötetyn diagrammin; se ei esimerkiksi tiedä, kuinka suurella alueella diagrammin sijainnit sijaitsevat. Se yksinkertaisesti sanottuna kertoo liikkua sijainnista toiseen, muttei kerro miten. (Red Blob Games, 2014)



Kuva 4. Esimerkkikuva AStar-algoritmin löytämästä nopeimmasta reitistä kahden eri sijainnin välillä (Red Blob Games, 2014).

Navigation2D-noodi toisaalta tarjoaa reitinhaun ja navigoinnin määritellyllä 2D-alueella NavigationPolygon-resurssien avulla. Oletuksena NavigationPolygon-resurssit lisätään automaattisesti niiden ilmentymien noodeista, mutta ne voidaan lisätä myös manuaalisesti. (Godot, n.d. g)

Unityn NavMesh-komponentti eli navigointiverkko mahdollistaa NavMeshAgent-komponentin käytön. NavMeshAgent-hahmot tunnistavat ja väistävät toisiansa sekä muita mahdollisia esteitä. Sen muokattaviin ominaisuuksiin kuuluu muun muassa säde, joka määrittää kuinka kaukaa se tunnistaa toiset agentit sekä törmäykset, pysähtymisetäisyys, joka määrittää, millä etäisyydellä se pysähtyy tavoitteestansa sekä nopeus, joka määrittää sen liikkumisnopeuden pelissä. (Unity Technologies, n.d. f)

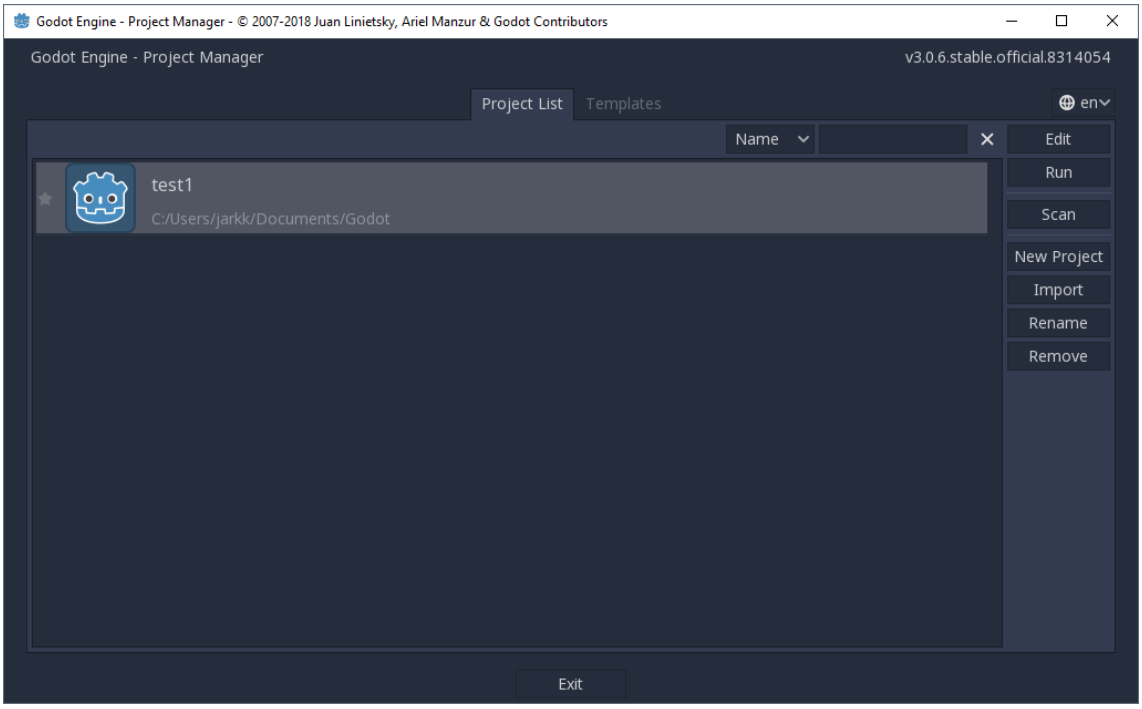
NavMeshAgent tarvitsee NavMesh-verkoston rakentamisen toimiakseen. Siinä tulee määrittää, mitkä ovat mahdollisia pintoja, joilla se voi kävellä sekä missä se ei voi kävellä. Jos NavMeshAgent-komponentti sijoitetaan hahmoon, joka ei sijaitse NavMesh-verkoston lähellä, se ei toimi. (Unity Technologies, n.d. g)

Godotin AStar- ja Navigation2D-noodin hyödyntäminen saattaa olla aloittelijalle haasteellista. AStar-noodi vaatii toimiakseen diagrammin, joka tulee luoda ja syöttää sille manuaalisesti. Navigation2D toisaalta kerää automaattisesti sallitun reitinetsintä alueen, mutta se vaatii silti tämän alueen hyödyntämistä manuaalisesti. Unityn NavMeshAgent- ja NavMesh-komponenttien käyttö on toisaalta erittäin yksinkertaista, mutta monimutkaisen NavMesh-verkoston luominen saattaa olla hankalaa.

Kaiken kaikkiaan Godotin reitinetsintä noodit vaikuttavat tehokkaammilta, mutta vaikeampi käyttöisemmiltä kuin Unityn ratkaisut. Siinä missä Godotin reitinetsintä ratkaisut vaativat manuaalista syöttötietoa, Unityn ratkaisut toimivat lähes oletusasetuksilla.

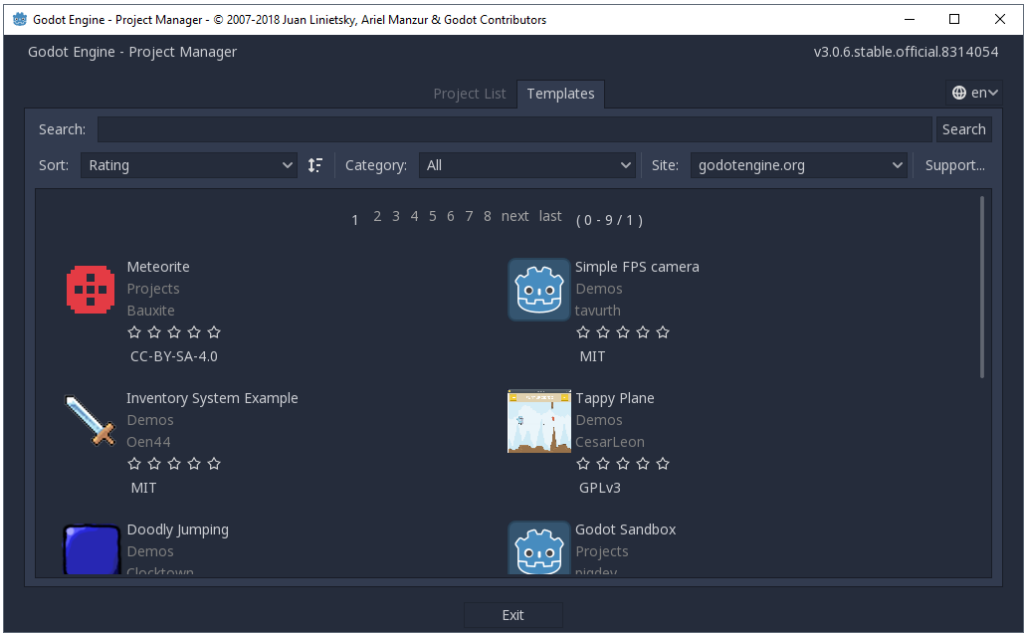
4 GODOT-EDITORIN KÄYTTÖ

Godotin käynnistäessä tulee Project Manager-ikkuna näkyviin, josta voidaan avata tai ajaa olemassa olevia projekteja. On mahdollista myös skannata kansioita olemassa olevien Godot-projektien löytämiseksi, jos ne eivät jostain syystä näy Project List-näkymässä. Täysin uuden projektin luomisen lisäksi on mahdollista tuoda projekti, joita voi löytää esimerkiksi Project List-näkymän viereisestä Templates-näkymästä. Projektin uudelleennimeäminen ja poistaminen ovat nopeita prosesseja ja mahdollisia myös Project Manager-näkymässä. Kuvassa 5 voidaan nähdä kaikki yllämainitut toimivuudet.



Kuva 5. Kuvankaappaus Godot-pelimoottorin Project Manager-näkymästä.

Templates-näkymästä voidaan ladata ja tuoda Godotin virallisia projekteja, demoja tai malleja. Nämä voivat auttaa aloittamaan projektin tekemisen tyhjästä, tai auttaa testaamaan Godotin toimivuuksia aloittelijalle. Virallisten projektien, demojen ja mallien lisäksi on mahdollista Support-napin kautta suodattaa Godot-yhteisön tekemiä projekteja, demoja ja malleja. Näkymästä voidaan nähdä myös näiden lisenssit. Templates-näkymän voi nähdä kuvasta 6.



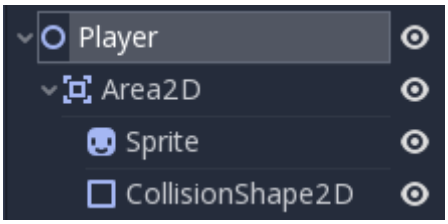
Kuva 6. Kuvankaappaus Project Manager-näkymän Templates-välilehdestä.

Godot projektin avatessa ylhäältä keskellä näkyy kuvassa 3 kuvattu Toolbar-palkki, josta voi vaihdella 2D-, 3D-, Script- ja AssetLib-näkymien välillä. 2D- ja 3D-näkymiä käytetään molempia pelintekemisessä riippumatta siitä, että onko tekemässä 2D- vai 3D-peliä. 2D-näkymää käytetään esimerkiksi 3D-peleissä käyttöliittymän luontiin, kun taas 3D-näkymää voidaan käyttää 2D-peleissä erottamaan tasojaan toisistaan. Esimerkiksi pelaajahahmon Z-akselin arvo on eri kuin background eli pelin taustan, jottei pelaajan sprite eli kuva uppoudu siihen, kun taas foreground eli pelin edustan Z-akselin arvo asetetaan pelaajahahmon edustalle. Tämä taas tekee sen, että pelaajahahmo pystyy menemään pelin edustan ”taakse”. Samankaltaisen mutta ei samanlaisen tehosteen saa aikaan layer- eli tasotekniikalla, jolla voidaan määrätä järjestys, missä eri mallit näkyvät, jos ne ovat päällekkäin. Esimerkiksi jos halutaan tehdä hiirtä seuraava liikkuva käsi, jossa on ase, voidaan tasotekniikalla määrätä, että käsi näkyy pelaajahahmon päällä ja ase näkyy halutusti joko käden alla tai päällä.

Script-näkymä toisaalta on Godotin sisäänrakennettu koodinmuokkausnäkymä. AssetLib on Unityn käyttäjille tutun Asset Store-näkymän kaltainen, josta käyttäjä voi ladata ja tuoda projektiinsa erilaisia hyödykkeitä, kuten pelaajahahmon mallin, tms. On kuitenkin huomattava, että näillä saattaa olla erilaisia lisenssiehtoja.

4.1 Noodit

Noodit ovat osia, joista peli koostuu. Niillä on nimi, ominaisuuksia ja kyky olla ali- tai yliluokkana toisille noodeille. Kun noodit toimivat toistensa yli- tai aliluokkina, niistä muodostuu noodipuu.

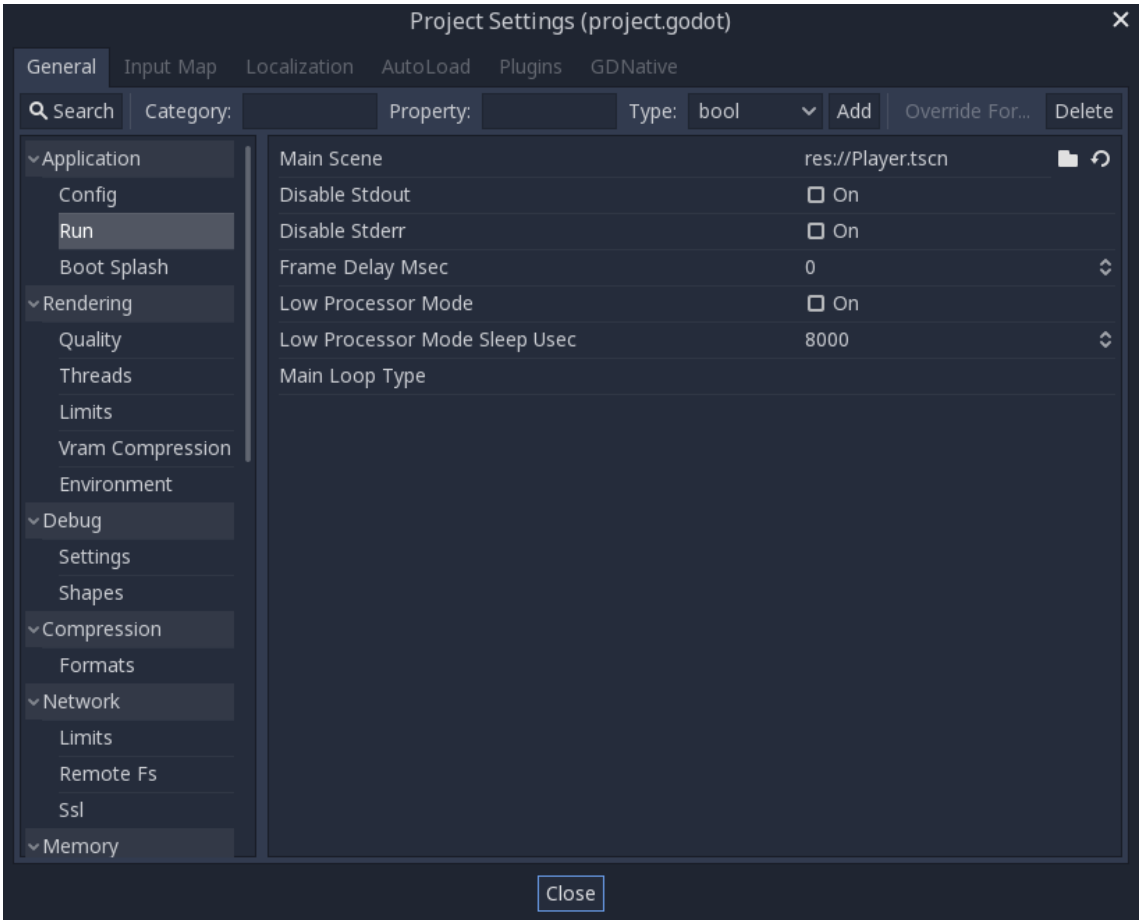


Kuva 7. Esimerkkitapaus yksinkertaisesta Player- tai pelaajahahmo-puusta 2D Godot projektissa. Player-noodi on tyhjä 2D-noodi, jonka aliluokkana toimii Area2D-noodi, jonka kahtena aliluokkana toisaalta toimii Sprite- ja CollisionShape2D-noodit.

Erilaisilla noodeilla on erilaisia toimintoja. Kuvassa 7 Player-puu koostuu neljästä erilaisesta noodista, joilla kaikilla on eri toiminto. Tyhjä 2D-noodi Player toimii säiliönä sen aliluokka noodeille. Area2D-noodi toimii törmäystunnistuksena sen aliluokka CollisionShape2D-noodille, joka edustaa törmäysmuototietoa, jota Area2D-noodi voi siten käyttää hyväksi. Sprite-noodi toisaalta edustaa pelaajahahmon mallia, joka on näkyvissä pelissä.

4.2 **Scene-järjestelmä**

Godot-projekti koostuu yhdestä tai useammasta Scene-osasesta, jotka toisaalta koostuvat noodiryhmistä. Godot-projekteissa tulee asettaa pää-Scene, jotta projektin voi ajaa. Kuten noodit, Scenen rakenneosasina toimivat noodit muodostavat puun kaltaisen hierarkian. Scene koostuu aina yhdestä juurinoodista, jossa muut noodit toimivat sen alinoodeina. Scene voidaan myös tallentaa ja ladata myöhemmin sekä instansoida, josta kerrotaan kappaleessa 4.3.

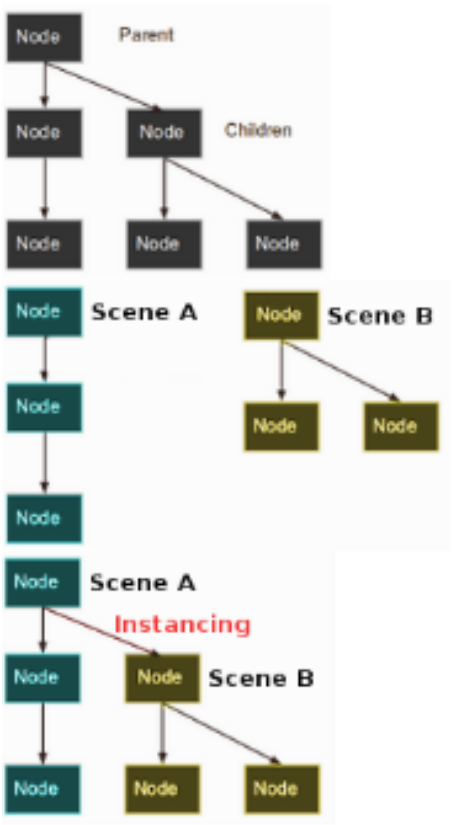


Kuva 8. Kuvankaappaus Project Settings-ikkunasta Godotin editorissa. General-välilehden Application-lehden Run-kohdan Main Scene-kohdasta voidaan vaihtaa projektin pää-scene. Jos pää-sceneä ei ole määritelty projektin asetuksissa, kun projektin ajaa, kysyy Godot, minkä scenen haluat asettaa pää-sceneksi, jotta kyseisen scenen voi ajaa.

4.3 **Instansointi**

Yhden scenen hallinta saattaa toimia pienissä projekteissa, mutta projektin kasvaessa koossa sekä vaikeudessa, voi silkka noodien määrä tulla vaikeaksi hallita. Tämän vuoksi Godot antaa mahdollisuuden erotella projekti eri sceneihin, joka auttaa järjestelemään projektin eri osat.

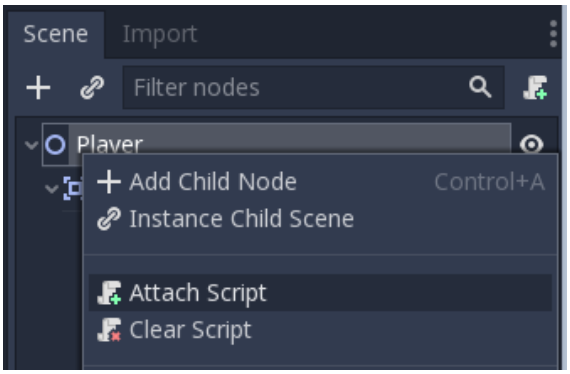
Kun yksittäisen scenen tallentaa, voi sen instansoida eli luoda ilmentymän siitä toiseen sceneen kuin se olisi mikä vain yksittäinen noodi. Alkuperäisen instanssin arvojen muokkaaminen vaikuttaa kaikkiin siitä instanssoitujen scenejen arvoihin. Yksittäisten instanssien arvojen muokkaus tapahtuu siinä scenessä, jossa halutaan arvon olla eri, kuin muissa sceneissä.



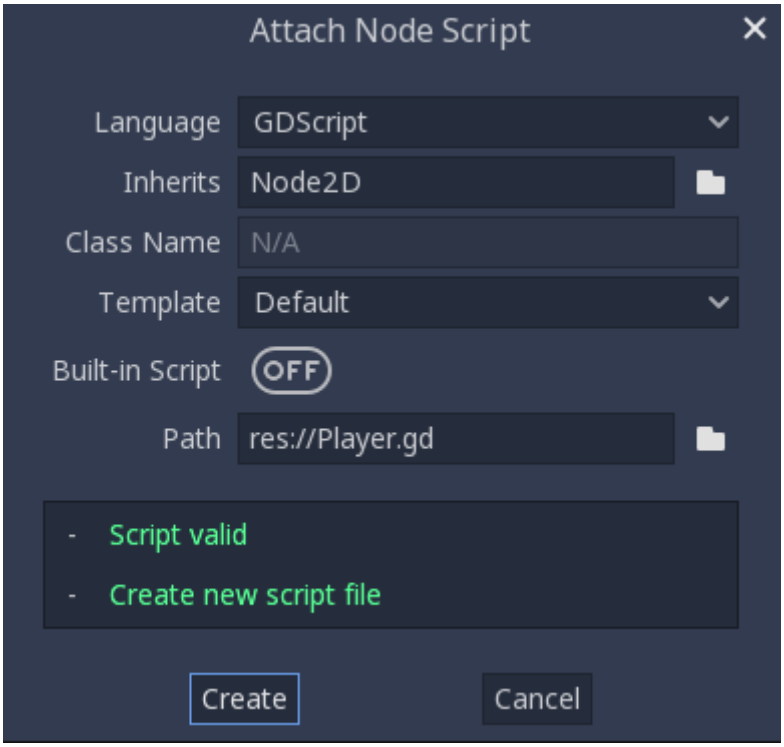
Kuva 9. Esimerkki noodien perinnästä sekä scenejen instansoinnista. Kuva n.d. Juan Linietsky, Ariel Manzur

4.4 Ohjelmointi Godotissa

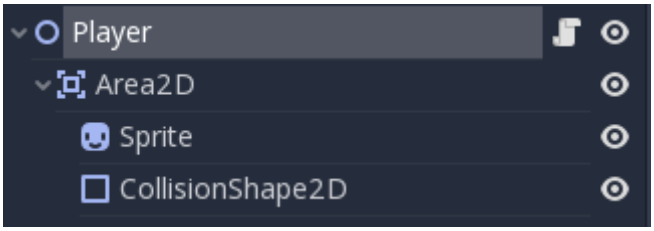
Skriptien eli komentosarjojen lisäys on olennaista pelintekemisessä. Niillä voidaan määrätä esimerkiksi; miten nopeasti pelaajahahmo voi liikkua, millä näppäimillä sitä ohjataan tai miten se reagoi, kun vihollishahmo tekee siihen vauriota. Komentosarjojen lisäys kuvataan kuvissa 10-12.



Kuva 10. Painamalla Attach Script-kohtaa voidaan lisätä haluttuun noodiin komentosarja. Oikean yläkulman lähellä oleva paperirulla, jossa on vihreä plussa ikoni, on Attach Script-toimivuuden pikanappi.



Kuva 11. Attach Script-valintaa painaessa avautuu seuraavanlainen ikkuna, josta voidaan asettaa komentosarjan ohjelmointikieli.



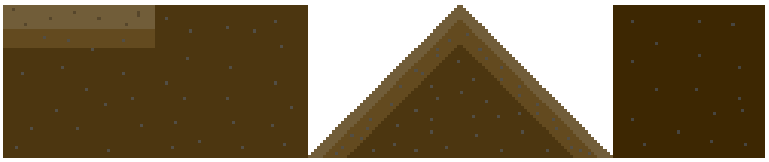
Kuva 12. Player-noodin oikeassa kulmassa näkyy paperirullaikoni, joka kuvaa sitä, että kyseisessä noodissa on komentosarja liitettynä. Painamalla tätä ikonia pystyy muokkaamaan kyseistä komentosarjaa.

5 PELITASOJEN LUONTI

Pelaajahahmo sekä vihollishahmot tarvitsevat ympäristön, jossa toimia. Tämän vuoksi pelitasojen luonti on tärkeä osa pelintekemistä.

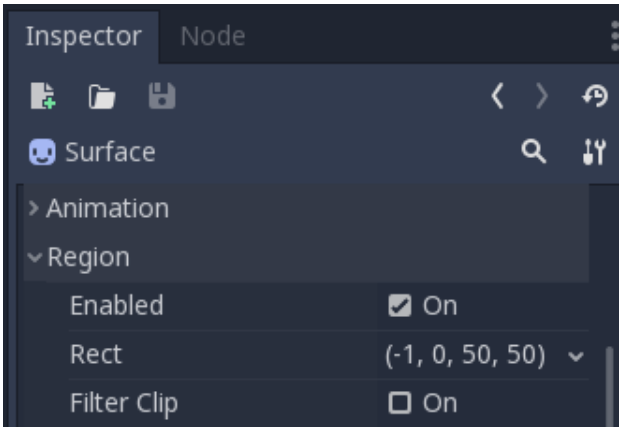
2D-tasohyppelyn pelitasojen perustalla on tasot eli englanniksi platforms. Näiden pelitasojen eli pelikenttien ideana on yleensä saattaa pelaajahahmo tasolta toiselle joko kulkemalla tai suoraan sanottuna tasohyppelemällä.

Pelikentät tarvitsevat rakennuspalikoita, joista pelikentät voidaan rakentaa. Tämän vuoksi luodaan tilemaps eli laattakartta.



Kuva 13. Pelissä käytettävä tileset eli laattaryhmä.

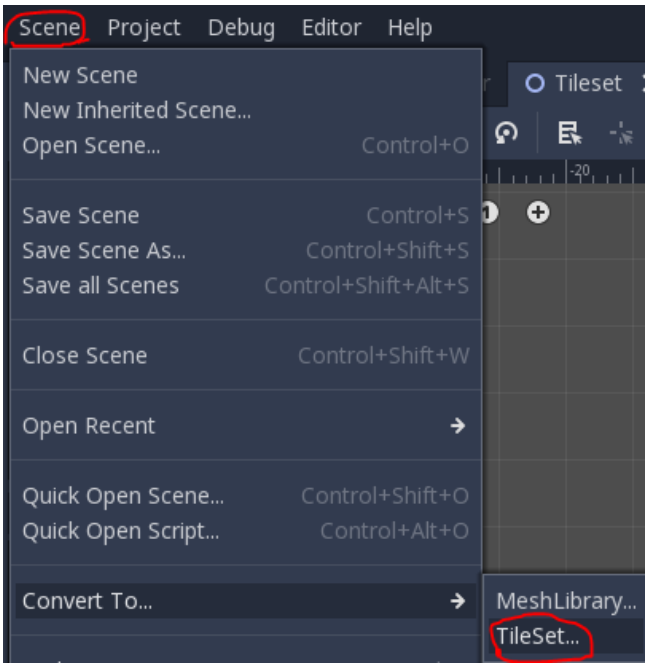
Laattakartan luonti tapahtuu Godotissa tuomalla mm. kuvan 13 mukainen laattaryhmä projektiin, josta muodostetaan laattakartta erottelemalla laattaryhmä yksittäisiin laattoihin. Erottelu tapahtuu Godotissa Inspector-välilehden Region-kohdassa.



Kuva 14. Esimerkkikuva yksittäisestä laatasta, joka on eroteltu Rect-ominaisuudella Region-kohdassa.

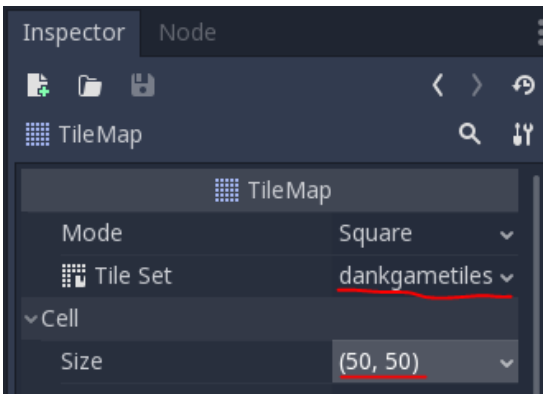
Laattojen erottelun jälkeen annetaan niille törmäystunnistukset sekä staattinen runko, joka pystyy tunnistamaan ja käyttämään 2D-fysiikoita pelissä. Yksinkertaisesti sanottuna nämä tekevät sen, että peli tunnistaa pelaajahahmon kävelyn niiden päällä ja estävät, ettei pelaajahahmo putoa niistä läpi.

Kun laattojen erottelu ja staattisten runkojen sekä törmäystunnistukset on lisätty, voidaan laatat tallentaa yksittäisenä scenenä. Tämän jälkeen sen voi muuntaa toimivaksi laattaryhmäksi.



Kuva 15. Kuinka scenen voi muuntaa TileSet eli laattaryhmäksi.

Laattaryhmän muuntamisen jälkeen luodaan uusi scene, johon lisätään juurinoodin alinoodiksi TileMap-noodi, jonka Tile Set-arvoksi Inspector-välilehdessä asetetaan aikaisemmin tallennettu laattaryhmä. Inspector-välilehden Cell-kohdasta vaihdetaan Size-ominaisuus yksittäisen laatan kokoa vastaavaksi.

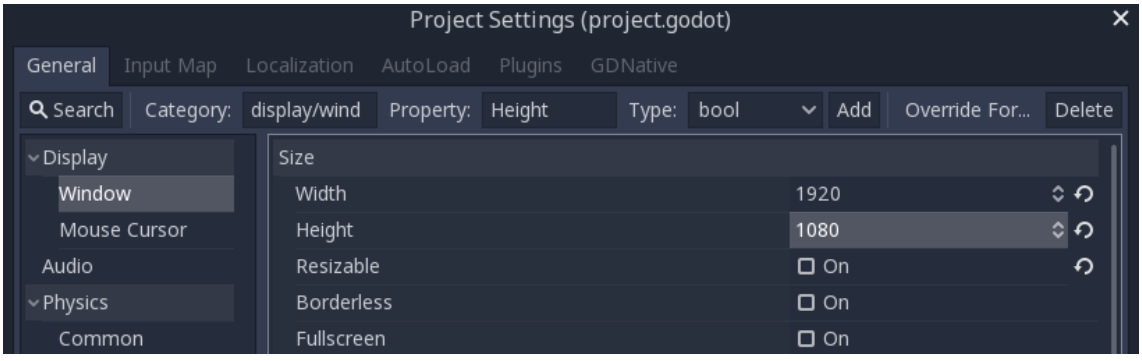


Kuva 16. Tile Set-ominaisuus asetetaan aikaisemmin tallennetun scenen mukaiseksi pudotusvalikosta, sekä Size-ominaisuus asetetaan tässä tapauksessa 50 molempiin X- sekä Y-arvoihin, sillä ne vastaavat yksittäisten laattojen alkuperäistä kokoa.

Kuva 17. Kuvankaappaus TileMap-noodin työkalupalkista. Yksittäisiä laattoja voi valita tästä valikosta sekä yläkulmassa olevista napeista voidaan kääntää laattojen suuntausta. Palkki tulee näkyviin, kun vastaavaa TileMap-noodia klikataan hiiren vasemmalla näppäimellä. Tämän työkalun avulla voidaan rakentaa halutun näköinen peliympäristö.

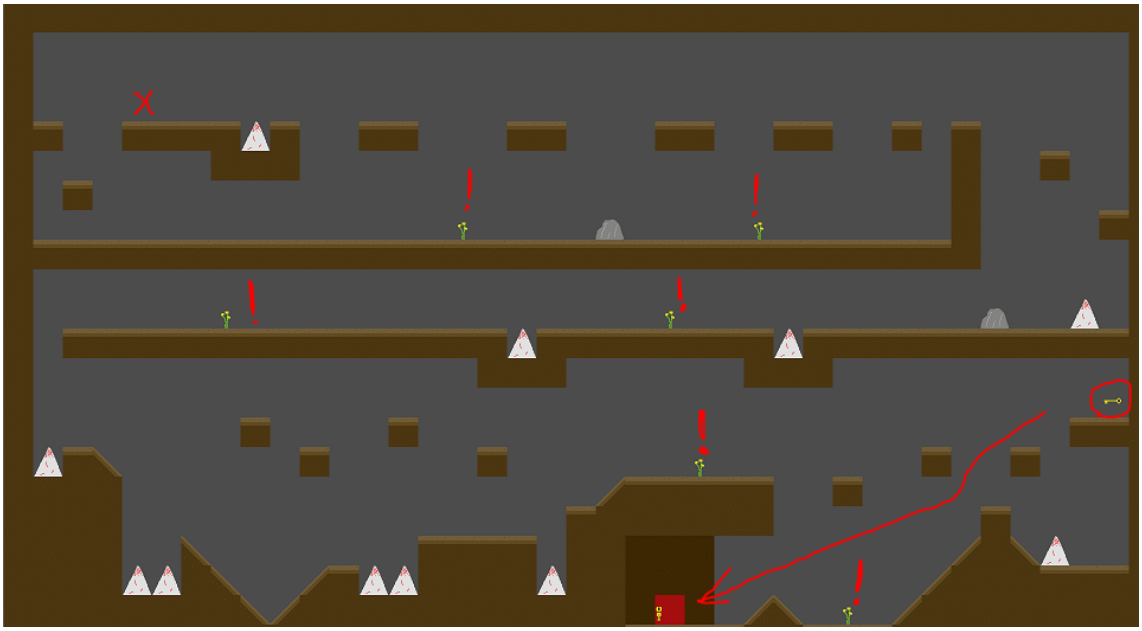
5.1 **Aloitustason luonti**

Ennen ensimmäisen tason luontia, voi olla hyödyllistä muuttaa viewport-ominaisuutta eli pelissä näkyvissä olevan ikkunan kokoa. Tämä tapahtuu Godotissa Project Settings-valikon General-välilehden Display-kohdan Windows-valinnasta. Oletusarvoina Width ja Height eli leveys- ja korkeusarvot ovat 1024 ja 600; tavoitteena on saada pelistä koko näytön kokoinen. Uusiksi arvoiksi asetetaan tässä projektissa 1920 ja 1080.



Kuva 18. Kuvankaappaus Project Settings-valikon kohdasta, josta pelin ikkunan kokoa voidaan muuttaa. On hyvä havaita alla olevat Resizable-, Borderless- ja Fullscreen-valintaruudut; näillä voidaan määrätä, voiko pelin ikkunan kokoa muuttaa, onko se rajaton tai onko se koko näytön kokoinen.

Kun ikkunan koko on halutun mukainen, voi pelikentän luonti alkaa. Aloitustasoon on lisätty hyppyesteitä, piikkiansoja sekä vihollishahmoja, jotka lähtevät seuraamaan pelaajahahmoa, jos se tulee liian lähelle niitä. Lisäksi ne ja piikkiansat lähettävät pelaajahahmon takaisin tason alkuun, jos ne koskevat pelaajaa. Tarkemmat tiedot vihollishahmoista kuvataan kappaleessa 7 Tekoälyn luonti.



Kuva 19. Kuvankaappaus aloitustasosta ilman pelaajahahmoa tai vihollishahmoja. Punainen X-merkintä vasemmassa yläkulmassa merkitsee, mistä pelaajahahmo aloittaa tason. Punaiset huutomerkit kukkien lähellä ovat merkkinä vihollishahmojen sijainnista. Avain, joka tarvitaan, että päästään seuraavaan tasoon on merkitty punaisella ympyrällä, ja avaimesta lähtevä nuoli kertoo avaimen avaavan oven sijainnin.

Ennen vihollishahmoja lisätään piikkiansoille komentosarja, jolla voidaan antaa niille toiminto. Ideana on tappaa pelaajahahmo, jos se osuu piikkiansaan. Komentosarjan alku saadaan automaattisesti tuotettua luomalla signaali Node-välilehdestä ja valitsemalla haluttu signaalitapa. Tähän tapaukseen käytämme body_entered-signaalia, sillä se osaa tunnistaa, kun sen törmäysalueelle tulee pelaajahahmo.

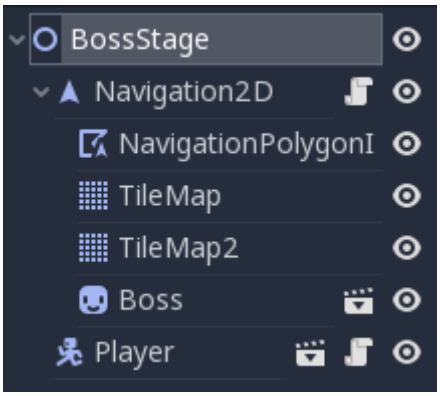
```
17 func _on_Area2D_body_entered(body):
18     if body.has_method("kill"):
19         body.kill()
```

Kuva 20. Piikkiansan komentosarja. Funktio ottaa vastaan siihen osuneita törmäyksiä, ja tarkastaa onko niillä ”kill”-metodi ja suorittaa sen, jos niillä on se. Tämä metodi tulee sijaitsemaan pelaajahahmolla, joka asettaa pelaajahahmon tilan kuolleeksi.

Lopetustasoon siirtyminen tapahtuu avaamalla aloitustasossa sijaitsevan oven avaimella, jonka pelaajahahmo voi poimia. Avaimen ja oven komentosarjat ovat samankaltaisia kuin piikkiansan. Niissäkin asetetaan signaali, ja viitataan pelaajan set_open_door- ja open_door-komentosarjoja. Ovi ei aukea, ellei pelaajahahmo nappaa avainta. Avaimen ollessa pelaajahahmon hallussa pääsee ovesta lopetustasoon.

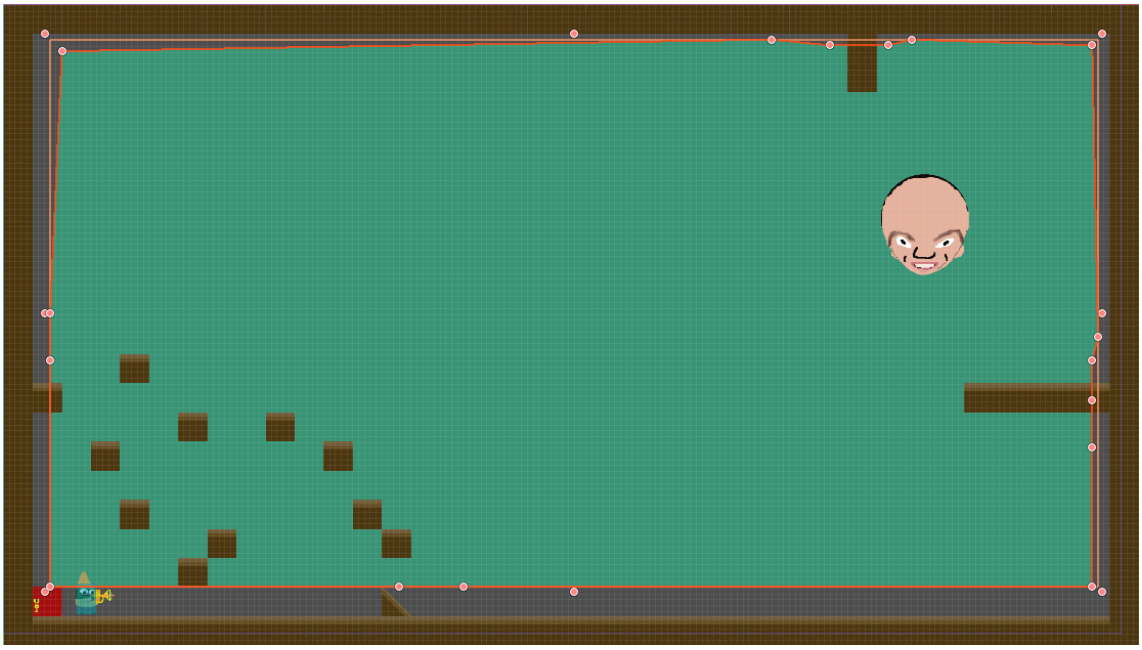
5.2 **Lopetustason luonti**

Lopetustasossa sijaitsee vihollishahmojen pomo. Toisin kuin aloitustasossa, vihollispomo tulee käyttämään AStar-reitinhaku algoritmin sijasta Navigation2D-navigointi ja reitinhakualgoritmia. Siinä missä AStar tarvitsee yhdistetyn verkon pisteitä, tarvitsee Navigation2D NavigationPolygonInstance-alueen. Sen voi luoda Navigation2D-noodin lisäämisen jälkeen sen alinoodiksi.



Kuva 21. Kuvankaappaus lopetustason noodihierarkiasta.

NavigationPolygonInstance-noodin lisäämisen jälkeen voidaan määrittää halutun muotoinen ja kokoinen monikulmioalue, jossa Navigation2D-noodi voi suorittaa reitinhakulogiikkansa. Kappaleessa 7.2 Tekoälyn luonti osuudessa käydään läpi Navigation2D-noodin funktioiden logiikka.



Kuva 22. Kuvankaappaus lopetustason NavigationPolygonInstance-alueesta. Vihreä alue määrittää alueen, jossa oikeassa yläkulmassa oleva vihollishahmo voi myöhemmin reitinhaku logiikan lisättyä liikkua.

6 KONTROLLIEN LUONTI

Jotta peliä voi pelata, tulee siinä olla jonkinlaiset kontrollit. Ilman minkäänlaista hallintaa pelinkulkuun peli ei ole pelattavissa, eikä eroa monessa tapauksessa esimerkiksi videosta.

Tulevissa kappaleissa kuvataan, kuinka Godotissa määritellään liikkumis-, hyppimis- ja hyökkäämiskontrollit komentosarjojen puolelta. Lopuksi käydään läpi, kuinka animaatio kytketään näihin kontrolleihin. Näitä ennen kuitenkin kerrotaan, kuinka Godotissa asetetaan mukautetut näppäimistökomennot.

Project Settings-valikon Input Map-välilehti sisältää oletuksena käyttöliittymän hallintaan tarkoitettuja komentoja. Näitä voitaisiin käyttää myös muihin kontrolleihin, mutta oppimisen vuoksi lisätään mukautetut kontrollit. Tämä tapahtuu kirjoittamalla yläpuolen Action-kohtaan halutun kontrollin nimi ja painamalla sen jälkeen Add-nappia, joka lisää kontrollin alla olevaan kontrollilistaan. Tämän jälkeen +-nappia painamalla voidaan valita haluttu näppäimistönnäppäin, ohjaintoiminto tai hiirennäppäin.

Kuva 23. Kuvankaappaus lisätyistä kontrolleista, joille on annettu mukautetut näppäimet. Vasemmalle pelaajahahmo liikkuu vasemmasta nuolinäppäimestä ja oikealle toisaalta oikeasta nuolinäppäimestä. Hyppy tapahtuu Z-näppäintä painamalla ja hyökkäys tapahtuu X-näppäimestä.

6.1 **Liikkuminen**

Komentosarjan puolella tulee nyt kytkeä kontrollit toimintoihin. Ensiksi kuitenkin lisätään vakioita sekä muuttujia, joita tarvitaan liikkumiseen, hyppimiseen, painovoimaan sekä lattiantunnistukseen.

```
6  const GRAVITY_VEC = Vector2(0, 900)
7  const FLOOR_NORMAL = Vector2(0, -1)
8  const SLOPE_SLIDE_STOP = 25.0
9  const MIN_ONAIR_TIME = 0.1
10 const WALK_SPEED = 300 # pixels/sec
11 const JUMP_SPEED = 450
12
13 var lv = Vector2()
14 var onair_time = 0 #
15 var on_floor = false
16 var is_dead = false
```

Kuva 24. Kuvankaappaus pelin alustavista vakioista sekä muuttujista.

Kuvassa 24 voidaan nähdä pelissä käytetyt vakiot ja muuttujat. Vakioina toimii mm. painovoimavektorivakio GRAVITY_VEC, joka määrittelee pelaajahahmon putoamisnopeuden Y-akselin arvon. FLOOR_NORMAL-vakio määrittelee, mikä on seinä, lattia tai katto. SLOPE_SLIDE_STOP-vakio estää pelaajahahmoa liukumasta kaltevia pintoja alas riippuen siitä, kuinka suureksi arvo määritellään. Pienemmillä arvoilla pelaajahahmo ei pysty pysymään paikallaan jyrkillä pinnoilla. MIN_ONAIR_TIME-vakiota käytetään lattiantunnistuksen yhteydessä. Jos arvo on suuri, pystyy hahmo hyppäämään ilmassa. WALK_SPEED ja JUMP_SPEED määrittelevät, kuinka monta pikseliä sekunnissa hahmo liikkuu tai hyppää ylöspäin. Muuttuja lv tarkoittaa linear_velocity-arvoa eli lineaarista nopeutta, jota käytetään painovoiman lisäyksessä, liikkumisessa sekä hyppimisessä. Muuttujia onair_time ja on_floor käytetään lattiantunnistuksen laskemisessa. Muuttuja is_dead toisaalta määrittelee, onko pelaajahahmo kuollut. Tämä voidaan liittää kontrolleihin; jos pelaajahahmon is_dead-arvo on tosi, se ei pysty liikkumaan.

Alustavien muuttujien ja vakioiden lisäyksen jälkeen voidaan niillä luoda logiikka komentosarjaan, jolla pelaajahahmo on vuorovaikutuksessa pelitason sekä kontrollien kanssa. Lähes kaikki fysiikkaan liittyvät prosessit tulevat funktion _physics_process sisälle, jota kutsutaan fysiikan prosessoinnissa (Godot, n.d. f).

Vaakasuuntaiseen liikkumiseen tarvitaan kohdenopeus-muuttuja, jota voidaan muuttaa riippuen siitä, mihin suuntaan pelaajahahmo liikkuu. Tämän jälkeen voidaan määritellä ehtolausekkeisiin aikaisemmin lisätyt kontrollit ja mahdolliset muut ehdot pelaajan liikkumiselle.

```
var target_speed = 0
if Input.is_action_pressed("left") && is_dead == false:
    target_speed += -1
if Input.is_action_pressed("right") && is_dead == false:
    target_speed += 1

target_speed *= WALK_SPEED
lv.x = lerp(lv.x, target_speed, 0.1)
```

Kuva 25. Kuvankaappaus pelaajahahmon vaakasuuntaisesta liikkumisesta, jossa määritellään pelaajan vasemmalle ja oikealle liikkuminen. Tämän lisäksi lineaarinen nopeus normalisoidaan lerp-funktiolla.

Näiden toimivuuksien lisäämisen jälkeen pelaajahahmolla voi liikkua vasemmalle sekä oikealle. Tasohyppely kuitenkin tarvitsee toimiakseen hyppimisen, sekä painovoiman, että pelaajahahmo pystyy putoamaan ja ylipäätänsä liikkumaan Y-akselilla.

6.2 **Hyppiminen**

Ensimmäiseksi lisätään lisälaskuri, joka synkronoi onair_time-muuttujan arvon delta-vakion kanssa, jonka avulla voidaan tasoittaa fysiikan prosessointi. Tämä tekee sen, että pelaajahahmo pystyy myöhemmin tunnistamaan lattian sulavasti. Koodin puolella tämä voidaan tehdä kuvan 26 mukaisesti.

```
onair_time += delta
if is_on_floor():
    onair_time = 0
```

Kuva 26. Kuvankaappaus pelaajahahmon komentosarjan onair_time-lisälaskurista sekä ehtolauseesta, jossa määritellään onair_time-muuttujan arvoksi 0, jos pelaajahahmo koskettaa lattiaa. Tämä mahdollistaa pelaajahahmon hyppimisen myöhemmin.

Painovoiman lisääminen vaatii laskutoimituksen, jossa lineaarisen nopeuden arvoksi määritellään deltan ja GRAVITY_VEC-vakion arvot. Tarkan laskutoimituksen näkee kuvasta 27.

```
lv += delta * GRAVITY_VEC
```

Kuva 27. Painovoiman käytön laskutoimitus. Lineaarinen nopeus lv = lv + delta * GRAVITY_VEC.

Nyt kun painovoima on pantu käytäntöön, voidaan hyppiminen lisätä. Tämä toimii lähes samalla tavalla kuin vaakasuuntaisen liikkumisen lisäys.

```
if on_floor && Input.is_action_just_pressed("jump") && is_dead == false:
    lv.y = -JUMP_SPEED
```

Kuva 28. Kuvankaappaus pelaajahahmon hyppytoiminnosta. Ehtolauseessa tarkastetaan, koskeeko pelaajahahmo lattiaa, onko hyppynäppäintä painettu sekä onko pelaajahahmo elossa. Jos ehdot täyttyvät, asetetaan lineaarisen nopeuden Y-akselin arvoksi negatiivinen JUMP_SPEED-vakio.

6.3 Hyökkääminen

Ammuksena pelaajahahmolle tässä tapauksessa luodaan nuotti, joka nimetään sceneksi nimeltä note. Ammukselle tarvitaan tapa, jolla se voi tunnistaa törmäyksiä, joten sille asetetaan juurinoodiksi RigidBody2D-luokka, jota 2D fysiikkamoottori voi kontrolloida. Alinoodeiksi lisätään Sprite, joka toimii ammuksen mallina ja CollisionShape2D, joka sallii törmäysten tunnistuksen. Lisäksi juurinoodille annetaan signaali, jota se voi kutsua törmäysten tunnistuksessa. Tämän avulla voidaan määrittää vuorovaikutustoiminto ammusten ja vihollisten välille.

```
18 func _on_note_body_entered(body):
19     if body.has_method("take_damage"):
20         body.call("take_damage")
```

Kuva 29. Automaattisesti luotu metodi signaalin lisäämisessä, jolle on määritelty piikkiansan kaltainen toiminto.

Nyt kun ammukset on luotu, tarvitaan tapa, jolla pelaajahahmo voi niitä käyttää. Tässä tapauksessa esiladataan ilmentymä nuotin scenestä, jonka jälkeen määritetään sijainti, johon se tuodaan. Sijainnin määrittelyn jälkeen määritetään sen nopeus ja lisätään poikkeus törmäyksentunnistukseen, jottei ammukset osu pelaajahahmoon itseensä. Tämän jälkeen ammuksen ilmentymä lisätään pelitasoon. Lopuksi soitetaan hyökkäysääni, joka soi hyökkäyksen yhteydessä sekä asetetaan shoot_time-vakion arvo nollaan. Kuvassa 30 voidaan nähdä tarkemmin hyökkäyslogiikan käyttö pelaajahahmon komentosarjassa.

```
76     if Input.is_action_just_pressed("attack") && is_dead == false:
77         var note = preload("res://note.tscn").instance()
78         note.position = $sprite/note_shoot.global_position
79         note.linear_velocity = Vector2(sprite.scale.x * NOTE_VELOCITY, 0)
80         note.add_collision_exception_with(self)
81         get_parent().add_child(note)
82         $trumpet_sound.play()
83         shoot_time = 0
```

Kuva 30. Kuvankaappaus pelaajahahmoon kytketyn komentosarjan hyökkäyslogiikasta.

6.4 Animaation kytkeminen kontrolleihin

Animaatiot ovat pelin sisäisten mallien muutoksia ja tiloja. Esimerkiksi pelaajahahmolla on useimmiten kävelyanimaatio sekä hyppäysanimaatio. Tässä työssä pelaajahahmolla on idle, jump ja walkcycle animaatiotilat, jotka ovat suomeksi toimeton, hyppy- ja kävelyanimaatiot.

Godotissa on mahdollista lisätä yksittäisen Sprite-noodin lisäksi myös AnimatedSprite-noodi, johon voidaan lisätä useita tekstuureja animaatioiden luomiseen. idle- ja jump-tiloissa käytetään vain yhtä kuvaa, kun taas walkcycle-tilassa käytetään viittä eri kuvaa, jotta kävelyanimaatio näyttäisi sulavammalta. Kuvassa 31 esitetään, miltä walkcycle-tila näyttää AnimationSprite-noodissa, kun se on lisätty.

Kuva 31. Kuvankaappaus walkcycle-animaatiotilasta.

Seuraavaksi tulee määritellä, miltä kävelyanimaatio näyttää. Tähän tarvitaan AnimationPlayer-noodia, joka mahdollistaa animaatioiden käytön juurinoodissa. AnimationPlayer-noodin lisäyksen jälkeen voidaan avata Animation-välilehti, jossa voidaan määritellä animaatioiden aikajana. Kävelyanimaation tapauksessa tulee asettaa kaikki 5 kuvaa aikajanalle haluttuihin kohtiin. Tässä tapauksessa asetetaan animaation kestoksi 1 sekunti, joten animaation alkuun laitetaan ensimmäinen kuva, jonka jälkeen seuraava kuva tulee 0,2 sekunnin jälkeen aikajanalla ja niin edelleen. Näin animaatio näyttää sutjakkaalta pelissä. Kuvassa 32 voidaan nähdä kävelyanimaation aikajana Godotin Animation-välilehdessä.

Kuva 32. Kuvankaappaus walkcycle-animaation aikajanasta. Valkoiset pisteet kertovat kuvien sijainnin aikajanalla.

Animaatioiden luonnin ja hiomisen jälkeen tulee ne kytkeä pelaajahahmon kontrolleihin. Tämä tapahtuu ohjelmointi puolella pelaajahahmon komentosarjassa.

Oletusanimaationa toimii idle- eli toimetontila. Hyppyanimaatio ja kävelyanimaatio kytketään on_floor-muuttujan tarkasteluun, joka tunnistaa, jos pelaajahahmo on lattialla. Hyppyanimaatiotila näytetään pelaajahahmolla, jos pelaajahahmo ei tunnista lattiaa ja sen lineaarisen kiihtyvyyden nopeus on pienempi kuin 0. Muuten pelaajahahmon animaatiotila on toimeton. Lopuksi määritetään ehto, että jos new_anim-muuttuja ei vastaa anim-muuttujaa, asetetaan sen arvoksi sama kuin new_anim-muuttuja, ja animaatiotila ajetaan. Muuten animaatio ei vaihdu, vaan pysyy samana. Kuvassa 33 kuvataan tarkemmin pelaajahahmon animaatioon liittyvä komentosarja.

```
89  if on_floor:
90      if lv.x < -CHANGE_SIDE_SPEED:
91          sprite.scale.x = -1
92          new_anim = "walkcycle"
93      if lv.x > CHANGE_SIDE_SPEED:
94          sprite.scale.x = 1
95          new_anim = "walkcycle"
96  else:
97      if Input.is_action_pressed("left") and not Input.is_action_pressed("right"):
98          sprite.scale.x = -1
99      if Input.is_action_pressed("right") and not Input.is_action_pressed("left"):
100         sprite.scale.x = 1
101 
102     if lv.y < 0:
103         new_anim = "jump"
104     else:
105         new_anim = "idle"
106 
107 if new_anim != anim:
108     anim = new_anim
109     $anim.play(anim)
```

Kuva 33. Kuvankaappaus pelin pelaajahahmon animaation kytkemisestä kontrolleihin. Ehtolauseen else-kohdassa pelaajahahmon kuva vaihtaa suuntaa riippuen, minne suuntaan pelaajahahmo on kävelemässä.

7 TEKOÄLYN LUONTI

Aikaisemmassa alaluvussa 3.6 vertailtiin ja kuvailtiin Unityn ja Godotin sisäänrakennettujen tekoälyjen käyttöä, ominaisuuksia sekä niiden toimintaa. Tulevissa kappaleissa toisaalta kerrotaan, kuinka Godotin AStar- sekä Navigation2D-luokkien kanssa luodaan toimivat reitinhakutekoälyt tähän projektiin.

7.1 AStar-luokan hyödyntäminen

AStar-reitinhakualgoritmi tarvitsee toimiakseen useita pisteitä, joiden väliltä se laskee nopeimman reitin aloituspisteestä lopetuspisteeseen. Tässä projektissa nämä pisteet voidaan liittää jo aiemmin käytettyyn laattakartastoon, jolla voidaan piirtää alue, jossa vihollishahmo pystyy liikkumaan ja siten pisteiden kautta löytämään pelaajahahmon sijainnin.

```
37 func _add_traversable_tiles(traversable_tiles):
38
39     # Looping all tiles
40     for tile in traversable_tiles:
41
42         # Get the tile's ID
43         var id = _get_id_for_point(tile)
44
45         # Add the tile to the A* navigation
46         astar.add_point(id, Vector3(tile.x, tile.y, 0))
```

Kuva 34. Kuvankaappaus komentosarjasta, joka lisää laataston AStar-verkostoon.

Kun käytettävä laatasto on lisätty AStar-verkostoon, tarvitsee se vielä komentosarjan, joka yhdistää laataston kaikki laatat toisiinsa niiden ympäriltään. Tätä varten sen tarvitsee käydä jokaisen kuvan 34 laatan läpi, sekä hakea niiden ID eli tunniste, jotta se tietää mikä laatta on missäkin. Tämän jälkeen se yhdistää jokaisen laatan sen naapureiden kanssa, luoden toimivan AStar-verkoston.

```
func _connect_traversable_tiles(traversable_tiles):

    # Loopin all tiles
    for tile in traversable_tiles:

        # Get the tile's ID
        var id = _get_id_for_point(tile)

        # Loops used to search around player (range(3) returns 0, 1, and 2)
        for x in range(3):
            for y in range(3):

                # Determines target, converting range variable to -1, 0, and 1
                var target = tile + Vector2(x - 1, y - 1)

                # Determines target ID
                var target_id = _get_id_for_point(target)

                # Do not connect if point is same or point does not exist on A*
                if tile == target or not astar.has_point(target_id):
                    continue

                # Connect points
                astar.connect_points(id, target_id, true)
```

Kuva 35. Kuvankaappaus AStar-verkon laattojen yhdistämisestä.

Lisäksi AStar tarvitsee toimiakseen itse reitinhaun logiikan; pelkkä verkoston kytkeminen ei riitä. Alla oleva kuva 36 esittää vihollishahmon käyttämää reitinhakua.

```
# Performed on each step
func _process(delta):

    # Only do stuff if we have a current path
    if path:

        # The next point is the first member of the path array
        var target = path[0]

        # Determine direction in which bee must move
        var direction = (target - position).normalized()

        # Move bee
        position += direction * MOVEMENT_SPEED * delta

        # If we have reached the point...
        if position.distance_to(target) < POINT_RADIUS:

            # Remove first path point
            path.remove(0)

            # If we have no points left, remove path
            if path.size() == 0:
                path = null
```

Kuva 36. Kuvankaappaus vihollishahmon reitinhaku logiikasta. Jos AStar on luonut reitin, vihollishahmo seuraa sitä. Komentosarja tarkastaa myös, onko vihollishahmo saapunut päämääräänsä. Jos vihollishahmo on saapunut päämääräänsä, poistaa komentosarja kyseisen reitin.

Ennen kuin vihollishahmo voi löytää pelaajan, tarvitsee se kuitenkin jonkin signaalin, joka käynnistää reitinhaun. Tämän voi luoda kytkemällä Godotin kautta signaalin pelaajahahmon liikkeisiin. Kun signaali on kytketty liikkeisiin, tulee se myös kytkeä vihollishahmoon ja sen AStar-verkostoon. Kuvassa 37 on kuvattu, kuinka kytkentä tapahtuu.

```
7  func _ready():
8  
9      player.connect("AGGRO", self, "_calculate_new_path")
10 
11 func _calculate_new_path():
12 
13     # find a path
14     var path = navigation_map.get_path(enemy_bee.position, player.position)
15 
16     # if we have a path to follow
17     if path:
18 
19         # remove enemy_bee's point
20         path.remove(0)
21 
22         # set the path
23         enemy_bee.path = path
```

Kuva 37. Kuvankaappaus reitinhaun aktivointisignaalin kytkemisestä pelaajahahmon ja vihollishahmon välillä. Pelaajahahmon signaali nimeltä AGGRO kytketään reitinhakuverkostoon. Komentosarjassa määritellään myös vihollishahmon aloitus- sekä lopetuspisteet sen reitinhaulle ja asetus reitille itselleen.

Kuvan 37 AGGRO-signaali lähetetään aina, kun pelaaja hyppää tai liikkuu mihinkä tahansa suuntaan. Tällä tavalla AStar päivittää sen reitinhakulogiikkansa pelaajan reaaliaikaisen sijainnin mukaan.

```
63     # Horizontal Movement
64     var target_speed = 0
65     if Input.is_action_pressed("left") && is_dead == false:
66         target_speed += -1
67         emit_signal("AGGRO")
68     if Input.is_action_pressed("right") && is_dead == false:
69         target_speed += 1
70         emit_signal("AGGRO")
```

Kuva 38. Kuvankaappaus pelaajahahmon päivitetystä liikkumislogiikasta. Godotin sisäänrakennettu emit_signal-komento lähettää luodun AGGRO-signaalin, joka aikaisemmin kytkettiin vihollishahmon reitinhakulogiikkaan.

7.2 Navigation2D-luokan hyödyntäminen

Navigation2D ei tarvitse erillistä verkostoa reitinhakuun, vaan sen sijaan se tarvitsee vain määritetyn alueen. Näin ollen sen reitinhakulogiikan rakentamiseen ei tarvitse luoda tai yhdistää sen toimialueen laattoja toisiinsa, tehden siitä helppokäyttöisemmän kuin AStar. Sen reitinhaku tarvitsee kuitenkin myös jonkinlaisen alkusignaalin, joka määrittää, milloin reitinhaku käynnistyy. Tässä projektissa alkusignaali kytkettiin pelaajahahmon hyppyihin, joka myös päivittää vihollishahmon reitinhaun loppupisteen.

```
func _input(event):
    if not event.is_action_pressed('jump'):
        return
    _update_navigation_path($Boss.position, player.position)
```

Kuva 39. Funktio, jolla Navigation2D päivittää sen reitinhaun.

Reitinhaun päivitykseen Navigation2D tarvitsee vain vihollishahmon sekä pelaajahahmon sen hetkisen sijainnin, jotta se osaa luoda mahdollisimman nopean reitin. Godotin sisäänrakennettu get_simple_path ottaa nämä sijainnit parametreinä ja palauttaa niiden välisen reitin.

```
func _update_navigation_path(start_position, end_position):
    path = get_simple_path(start_position, end_position, true)
    path.remove(0)
    set_process(true)
```

Kuva 40. Navigation2D-luokan sisäinen reitinhaun päivitys.

Näiden lisäksi tulee lisätä viimepiste, missä vihollishahmo oli ja asettaa se vihollishahmon alkupisteeksi. Näin reitinhaun päivitys toimii jatkuvasti. Se tarvitsee myös silmukan, jolla se tarkastelee missä kohtaa reittiä se on ja reitin lopussa poistaa reitin, jotta se ei käyttäisi samaa reittiä uudelleen. Lopuksi reitinhakua kutsutaan _process(delta)-funktiossa, jota kutsutaan pelin pyöriessä jatkuvasti, luoden täten jatkuvan reitinhaku prosessin.

```
func _process(delta):
    var move_distance = boss_speed * delta
    pathmoving(move_distance)

func pathmoving(distance):
    var last_point = $Boss.position
    while path.size():
        var distance_between_points = last_point.distance_to(path[0])

        if distance <= distance_between_points:
            $Boss.position = last_point.linear_interpolate(path[0], distance / distance_between_points)
            return

        distance -= distance_between_points
        last_point = path[0]
        path.remove(0)

    $Boss.position = last_point
    set_process(false)
```

Kuva 41. Kuvankaappaus Navigation2D-luokan reitinhaun välimatkojen tarkastelusta. Lopuksi päämäärään saavuttuaan vihollishahmon reitinhaku päättyy.

8 YHTEENVETO

Työn tavoitteena oli tutkia Godotin pääpiirteitä ja ominaisuuksia ja vertailla niitä Unityn pääpiirteiden ja ominaisuuksien kanssa. Vertailuissa tuotiin ilmi Godotin ja Unityn yleiset eroavaisuudet ja jälkeenpäin pohdittiin niiden eri käyttötarkoituksia. Suurimmat vertailussa ilmi tulleet eroavaisuudet olivat muun muassa ryhmätyöskentelyominaisuudet, muokkausohjelman tiedostokoot, tuettujen ohjelmointikielien määrä sekä saatavilla oleva materiaalin määrä. Tämän lisäksi kuvattiin Godotin muokkausohjelman käyttöä, laadittiin kontrollit pelaajahahmolle luotuun peliin, yhdistettiin niihin animaatiot sekä käytettiin AStar- ja Navigation2D-reitinhakualgoritmejä hyväksi vihollishahmojen tekoälyn luonnissa. Päämääränä oli saada aikaan toimiva yksinkertainen peli, jotta mahdollisimman montaa Godotin ominaisuutta päästiin testaamaan. Toiminnallisuuksiltaan Godot toimi logiikaltaan samankaltaisesti kuin Unity, mutta erilaisella käyttöliittymällä.

Peliä luodessa käytettiin Godotin sisäistä GDScript-ohjelmointikieltä, sillä C#-ohjelmointikielen muokkausohjelma oli vielä kokeiluversiossa työn alussa. Työn loppuvaiheissa kuitenkin Godot korjasi C#-kokeiluversion valmiiksi versioksi, joka olisi mahdollisesti helpottanut työn tekoa, sillä C#-ohjelmointikieli on Unityssä tällä hetkellä ainoa täysin tuettu kieli. Siitä huolimatta työ sujui GDScript-kielellä lähes ongelmitta.

Työn tekemisessä onnistuttiin. Tutkimuskysymyksiin vastattiin sujuvasti. Lopputuloksena syntyi Godotin ja Unityn välinen vertailu, Godotin muokkausohjelman käyttöohjeet sekä yksinkertainen 2D-tasohyppelypelin luonnin dokumentointi kontrollien luonnista pelitasojen luontiin sekä tekoälyn hyödyntäminen AStar- sekä Navigation2D-reitinhakualgoritmien kautta.

Jatkotoimenpiteitä ei ole suunniteltu tälle opinnäytetyölle. Se kuvaa Godotin ominaisuuksia hyvin Unity-pelimoottorin käyttäjille, joka oli myös tämän työn pyrkimys. Mahdollinen jatkotoimenpide voisi kuitenkin olla Unreal Engine-pelimoottorin lisääminen vertailuun.

LÄHTEET

80 Level. (2016). Godot 2.0: Talking with the Creator. Blogijulkaisu 4.3.2016. Haettu 10.2.2019 osoitteesta https://80.lv/articles/godot2-interview/

Bastiaan, O. (2017). Godot 3's VR and AR support. Blogijulkaisu 6.11.2017. Haettu 1.2.2019 osoitteesta https://godotengine.org/article/godot-3-vr-and-ar-support

Blender Foundation. (n.d.). Blender 2.79 Manual. Haettu 1.2.2019 osoitteesta https://docs.blender.org/manual/en/latest/game_engine/physics/introduction.html

Brasseur, V. (2016). Godot open source game engine helps power the future in West Virginia. Blogijulkaisu 16.8.2016. Haettu 14.2.2019 osoitteesta https://opensource.com/education/16/8/godot-open-source-game-engine

Etcheverry, I. (2017). Introducing C# In Godot. Blogijulkaisu 21.10.2017. Haettu 2.2.2019 osoitteesta https://godotengine.org/article/introducing-csharp-godot

Fine, R. (2017). UnityScript's long ride off into the sunset. Blogijulkaisu 11.8.2017. Haettu 2.2.2019 osoitteesta https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/

Godot. (n.d.). a). Download. Windows. Haettu 1.2.2019 osoitteesta https://godotengine.org/download/windows

Godot. (n.d.). b). Features. Haettu 17.1.2019 osoitteesta https://godotengine.org/features

Godot. (n.d.). c). From Unity3D to Godot Engine. Haettu 10.02.2019 osoitteesta https://docs.godotengine.org/en/3.0/getting_started/editor/unity_to_godot.html

Godot. (n.d.). d). Godot Engine – Multi-platform 2D and 3D game engine. Haettu 10.02.2019 osoitteesta https://github.com/godotengine/godot

Godot. (n.d.). e). License. Haettu 15.1.2019 osoitteesta
https://godotengine.org/license

Godot. (n.d.). f). Node. Haettu 03.03.2019 osoitteesta
https://docs.godotengine.org/en/3.0/classes/class_node.html#class-node-physics-process

Godot. (n.d.). g). Navigation2D. Haettu 11.3.2019 osoitteesta
https://docs.godotengine.org/en/3.0/classes/class_navigation2d.html

Manzur, A. (2016). Our point'n'click framework is finally out! Blogijulkaisu 1.10.2016. Haettu 2.2.2019 osoitteesta
https://godotengine.org/article/our-point-click-framework-finally-out

natosha-bard. (2015). Unity on Linux: Release Notes and Known Issues. Blogijulkaisu 26.8.2015. Haettu 10.2.2019 osoitteesta
https://forum.unity.com/threads/unity-on-linux-release-notes-and-known-issues.350256/

Pranckevičius, A. (2018). Releasing the Unity C# source code. Blogijulkaisu 26.3.2018. Haettu 10.2.2019 osoitteesta
https://blogs.unity3d.com/2018/03/26/releasing-the-unity-c-source-code/

Red Blob Games. (2014). Introduction to the A* Algorithm. Blogijulkaisu 26.5.2014. Haettu 12.3.2019 osoitteesta
https://www.redblobgames.com/pathfinding/a-star/introduction.html

Software Freedom Conservancy. (n.d.). Current Projects. Haettu 14.2.2019 osoitteesta
https://sfconservancy.org/projects/current/

Suckley, M. (2016). OKAM Studio on empowering designers with Godot Engine's adventure game framework Escoria. Blogijulkaisu 15.8.2016. Haettu 2.2.2019 osoitteesta
https://www.pocketgamer.biz/news/63746/empowering-designers-with-adventure-framework-escoria/

Unity Technologies. (n.d.). a). Creating and Using Scripts. Haettu 17.2.2019 osoitteesta
https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html

Unity Technologies. (n.d.). b). The world's leading real-time creation platform. Haettu 17.2.2019 osoitteesta
https://unity3d.com/unity

Unity Technologies. (n.d.). c). Unity Store. Haettu 17.2.2019 osoitteesta https://store.unity.com/

Unity Technologies. (n.d.). d). Unity Teams. Haettu 17.2.2019 osoitteesta https://unity3d.com/teams

Unity Technologies. (n.d.). e). Navigation2D (Pathfinding for 2D Games). Haettu 11.3.2019 osoitteesta https://assetstore.unity.com/packages/tools/ai/navigation2d-pathfinding-for-2d-games-35803

Unity Technologies. (n.d.). f). NavMesh Agent. Haettu 11.3.2019 osoitteesta https://docs.unity3d.com/Manual/class-NavMeshAgent.html

Unity Technologies. (n.d.). g). Building a NavMesh. Haettu 11.3.2019 osoitteesta https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html

LIITTEET

Liite 1
Godotin ja Unityn erot (Godot, n.d. c).

	Unity	Godot
License	Proprietary, closed, free license with revenue caps and usage restrictions	MIT license, free and fully open source without any restriction
OS (editor)	Windows, macOS, Linux (unofficial and unsupported)	Windows, macOS, X11 (Linux, *BSD)
OS (export)	• **Desktop:** Windows, macOS, Linux • **Mobile:** Android, iOS, Windows Phone, Tizen • **Web:** WebAssembly or asm.js • **Consoles:** PS4, PS Vita, Xbox One, Xbox 360, Wii U, Nintendo 3DS • **VR:** Oculus Rift, SteamVR, Google Cardboard, Playstation VR, Gear VR, HoloLens • **TV:** Android TV, Samsung SMART TV, tvOS	• **Desktop:** Windows, macOS, X11 • **Mobile:** Android, iOS • **Web:** WebAssembly • **Console:** See Console Support in Godot • **VR:** Oculus Rift, SteamVR
Scene system	• Component/Scene (GameObject > Component) • Prefabs	Scene tree and nodes, allowing scenes to be nested and/or inherit other scenes
Third-party tools	Visual Studio or VS Code	• External editors are possible • Android SDK for Android export
Killer features	• Huge community • Large assets store	• Scene System • Animation Pipeline • Easy to write Shaders • Debug on Device